Natural Language Processing
Prof. Amrith Krishna
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 60
Tutorial

Hello everyone.

(Refer Slide Time: 00:21)

So, welcome to the tutorial session about word vectors or word embeddings. So, we will see how we can use word embeddings to find out relation between words or how we can find out again sentence representations or document representation using this word vectors. Fascinatedly aware that we use the distribution in semantic hypothesis that is a word can be defined by the words associated with it in it is context. So, we use this notion to build up representations for word and then we (Refer Time: 00:58) similar words using this particular hypothesis. So, for this task what we will be doing is that, we will be again using the same documents the review documents that we have used in the previous tutorial.

(Refer Slide Time: 01:24)

So, I have collected all of them together into a single csv document here and we use a data since library pandas that helps us in speedy processing of these or more efficient processing of these documents.

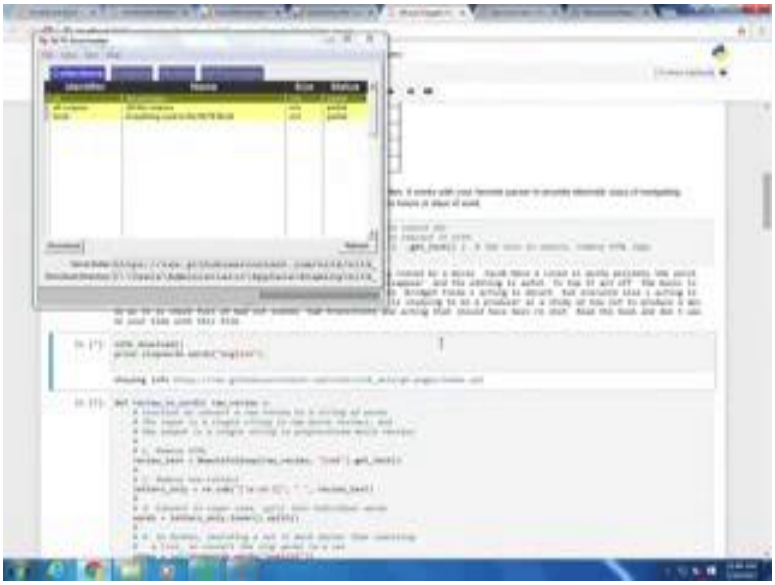
(Refer Slide Time: 01:31)

So, here we load all these documents into the data frame train. See if you find train

essentially has about 5 3 8 7 documents, which are those documents, which are like compilation of both positive and negative documents. And you have previously seen those 5 samples from this document collection. So, as we were mentioning in the previous tutorial that a lot of preprocessing needs to be done before using this particular document set.

For example, it has some extra html tags like br slash it has some other external markers. So, we will be removing the and we replace it with other normal characters. So, here this regular expression essentially looks for those entities which are in alpha and numeric. And now what we do is that we will also do stop word removal for this purpose we will be using the stop words that are stored inside nltk. See you can find nltk corpus we have imported the stop words. So, you might not have installed this by default in your systems.

(Refer Slide Time: 03:18)



So, what you can do is you can call the function nltk download from the notebook itself. It will show a pop up box where you can go and look for stop words package, see you can just select on one of those packages say a stop words and click on the download button, we already have it downloaded. Similarly, we will be using the tokenizer which is the (Refer Time: 03:41) token tokenizer that also should be downloaded before this notebook has to be used.

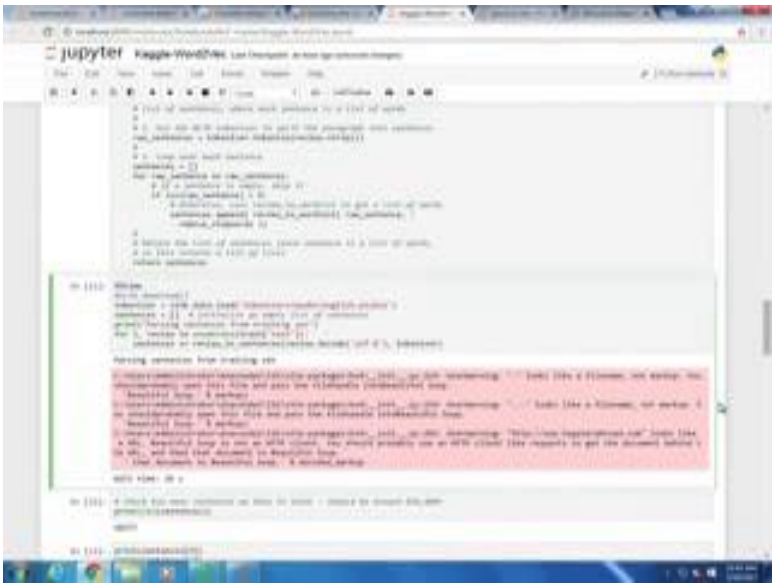
So, we take the stop words. So, we I will be commenting it out yeah and we find these are the stop words. Once we have this, what we do is that we convert each review and we extract each individual word from those reviews. And in that process we do the case folding and we remove the stop words and all other necessary preprocessing. So, we will be doing it for all of those sentences once we have it.

(Refer Slide Time: 04:30)

We need to have a suitable vector representation or a list representation initially and then convert it into a vector using the word embedding.

Yes, now we will be using the word to an embedding method. So, I assume you remember the word2vec embedding method that is taught in the class. Otherwise I recommend you to go through the lectures and see how exactly word2vec functions.

(Refer Slide Time: 05:05)



Unfortunately, we have genism package that provides us the word to a pre implemented in in python. So, once we convert this sentence, we will be training the model. So, in Jupiter notebook you can use magic functions or magic keywords, where if you use something like time, it will tell how much time if it requires to execute a particular function.

So, from all the 5000 documents we have extracted about 58,000 different sentences. So, we can have a look into how the sentence looks like. So, the first sentence looks like this movie is a perfect example of an excellent book getting ruined by a movie. So, there are other reviews as well.

(Refer Slide Time: 06:18)

Now, we have to convert each of this into a word vector. So, for each of this word we have to get an efficient vectorial representation or a vectorial representation.

So, what we do is that we have to set some parameters before we use this model.

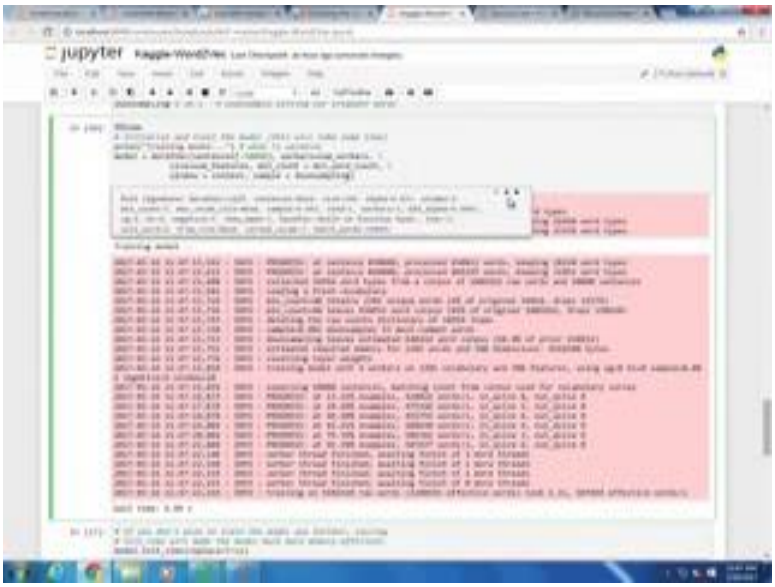
(Refer Slide Time: 06:38)

So, one thing is that we have to find how many or what is the dimension that the vector should have once word to a glance the vectors. So, we expect it to be somewhere around 300. So, this is often set empirically after trying out various dimension sizes. So, here output vectors all are for each word will be of size 300 now minimum words count. So, minimum words count is basically the number of times a word has to appear in the corpus to be considered for the vector formation, may be for this practical purpose I for this tutorial purpose I will be keeping it a bit lower.

So, this is a programming convenience is that what are the number of parallel threads to run this makes you a job faster, but it depends on the ability of your system. Next comes context window size. So, context window size essentially looks for given a sentence how many near nearby words have to be considered. So, if I consider the word example how many words left to this given word and right to the given word should be considered while forming the vector for this word. So, as we know distribution semantics depends on the word and it is nearby context and we here set a cap on, or a limit on the number of word that can appear in it is context. Whatever appears we need this count will be ignored by the model when we prepare the vector for each word.

So, though we have about 58 sentences in the interest of time, I will be using a less number of sentences here. So, we use the word to work model. This word to work model is defined in genism. So, that we can see that from genism I am importing word2vec.

(Refer Slide Time: 08:49)



So, once we have word2vec we keep it size to be 300, which is numb features and we keep then min word count the number of words that should appear. I mean the number of times it should appear before in in appear in a corpus because it needs to be considered.

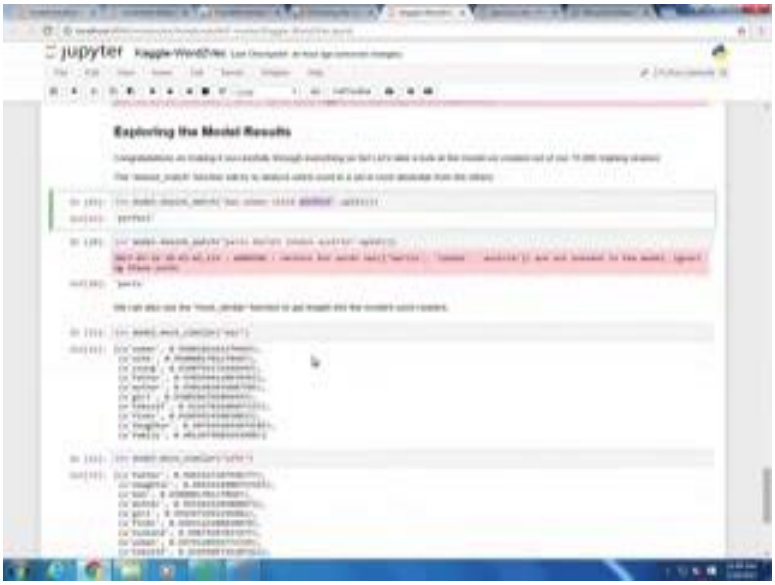
So, yeah when it comes ignore all those words with the total frequency lower than that particular count. So, you can use the shift tab keys in your keyboard to see the definition for each of these functions. Now there is this parameter sg equal to 0. So, if you remember there were 2 ways of using word2vec one was the skip gram model and the other was the cbow model. So, when we keep sg equal to 0 cbow is used, but if we use sg equal to one the system will or the model will use skip gram model. So, here by default we are keeping sg equal to 0. See we will train the model and let us see how much time it takes. So, in around 3 seconds we have our model trained.

So, let us try to increase the count of sentences and see how much more time it takes. See it takes almost double the size. So, there are some other arguments that you need to go through for example, there we give a down sampling as the argument to the as the value to the argument sample. So, it essentially says that if you have a certain number of words that is much more than what we require they are randomly round sampled. Similarly, here negative. So, when we use this function with this argument negative it basically takes

care of the negative sampling that word to a efficiently uses.

So, by default it looks for 5 negative samples. It can be set to varying values that will naturally effect the quality of the vectors that are generated.

(Refer Slide Time: 11:39)



So, now what we do is that we, once we have the model where we have a vector for each of those words which are there in 50,000 sentences what we do is that we give a new sentence and see which of these words are a most dissimilar to that in the model. So, we can find that kitchen is somehow not present in the word. So, it is ignored. So, amongst man woman and child, here child is the most dissimilar word from the other 2 words that is man and woman. For example, let us see some other word.

Say let us take one of those words like perfect and see is this change the result. So, now, amongst the 4 words man women child and perfect, you find that perfect is the most dissimilar to the other 3. Now we can also have this function most similar which basically tells given a word what are the other similar words in the entire corpus. So, we can find the word man has other similar words which are women killer father etcetera with it is corresponding similarity score. The similarity score that typically used is causing similarity though other similarity functions can also be efficiently used.

So, now if we find the word wife, the similar word that appears with the son father brother girlfriend with it is different similarity scores. So, though we cannot tell the exact relation between the words that is in the list, what we can find here is that the words that appear similar to this word are in reality similar to the given word. So, if we give the word aw full we find other similar words that can be used in same context as the word awful can be used we can find terrible horrible boring dull etcetera. So, this is essentially how we can use word2vec to find similar words.

(Refer Slide Time: 14:13)

So, you can have a look in to the Kaggle challenges where they have different competitions where we can use the word vectors. So, we have taken this tutorial from the Kaggle tutorial or the Kaggle competition called when a bag of words meet bag of popcorn, where you are supposed to build a task where you basically use the review dataset to build word vectors and then use them for the sentiment analysis task that we have previously seen.

(Refer Slide Time: 14:47)

So, they have different parts to this particular tutorial. It will be advisable to go through each of this. In addition I will be just referring to the doc2vec which is an improvement our word2vec.

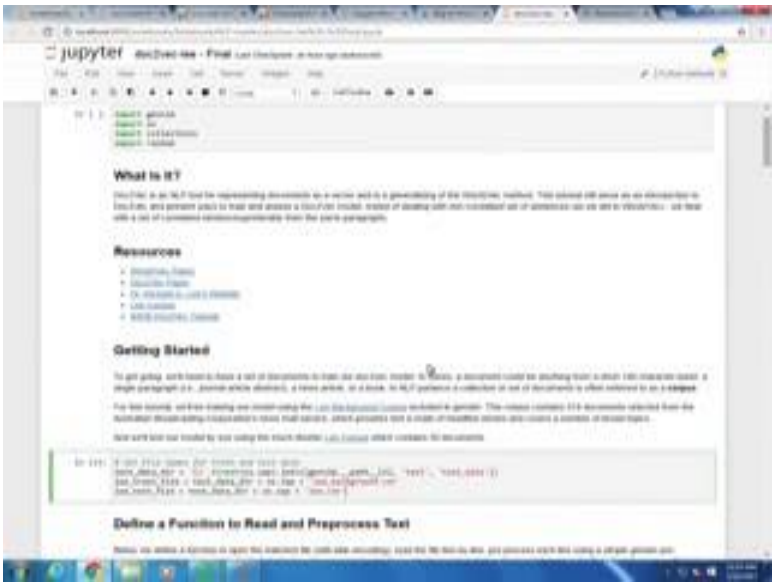
(Refer Slide Time: 15:04)

Or which is basically it cannot be set as an improvement, but it is a method to create

vector representation for the document. See we have seen in the topic modeling tutorial that we had representation vector representation for each document in this particular tutorial, we were not using any document representation we were using a vector for each individual words.

Now, how can we use this word2vec vectors to arrive at a document vector representation and we have this doc2vec method. So, you can look in to the doc2vec paper here which is linked to this particular notebook. Here we will be using the default tutorial provided in genism.

(Refer Slide Time: 16:01)



So, genism already has a corpus and it already has the doc2vec implemented in this model. So, we can just import the corpus that is stored in genism and we can find that they provided tagged corpus.

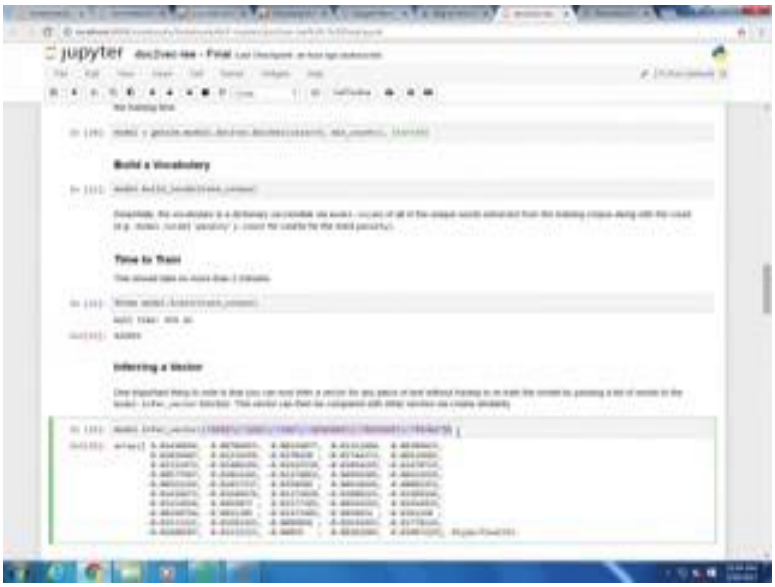
(Refer Slide Time: 16:20)

So, here is the corpus with individual words that forms the document and it is corresponding tag is given.

Similarly, there is another document with it is tag given as one now we have a test corpus. So, the test corpus the tag is not provided. Now we instantiate the doc2vec model, so as in the word2vec you can find different arguments that you can give to the doc2vec so here size is the dimensional vector of the feature vector. So, now, each document will be represented as a vector with the given size and you can look in to other arguments as well.

Now, once we have this model we build the vocabulary and we use it to train the corpus.

(Refer Slide Time: 17:24)



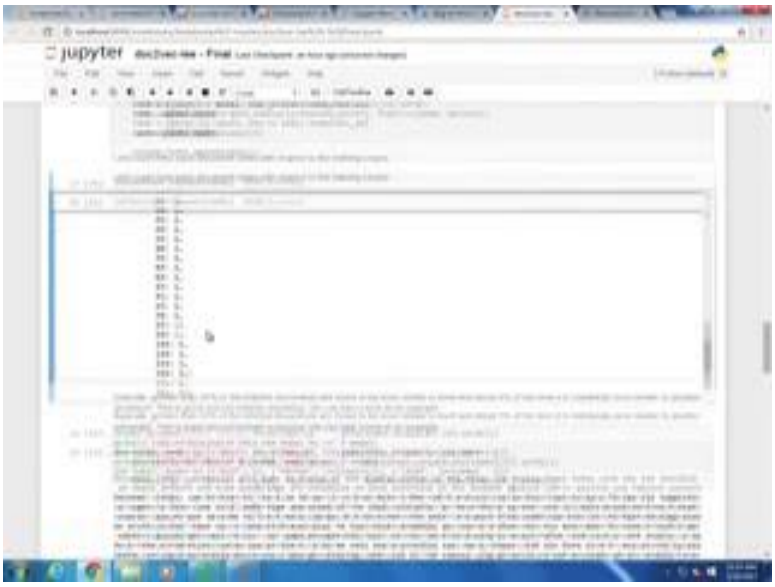
Now, with this model when you have this vector, that has the following words only you can prevent forest fires, we get a representation for that particular document. We can see what is it is size. So, we can find that the size of this particular vector is of 50.

(Refer Slide Time: 17:51)

Now, once we have this vector for each of the document in our training corpus what we

do is, that we take a particular document and we find it similarities with other vectors or other documents and we then form a rank for each of them. Once we have this once we have this computer the similarities. What we have done is that we have taken the vectors for each different document and we have compared amongst themselves and we find the similarity between those documents.

(Refer Slide Time: 18:28)



So, what do you find here is basically the similarity of a given document and it is rank. So, lower the number it is better and we can find that lot of them have very high similarity and they have they share the same value hence are ranked at 1. So, let us see which are those documents.

(Refer Slide Time: 18:54)

So, we were comparing this document and we find that the most similar and dissimilar documents are these documents. So, we can find the document 299 has the highest similarity or one of those with the highest similarities 83 and you can find you can have a look in to these 2 documents.

(Refer Slide Time: 19:14)

So, essentially if you see there are the contents in the both the document, basically talks about very similar manner very similar issues and hence they have similar words in the context. And thereby the model is able to find the document similarity. Here we find another document which is at the median. So, we can find that the similarities called between those at the highly ranked and the median are not much different. Hence still we can find the difference in the words that are used in context of all these documents.

(Refer Slide Time: 19:56)

And the least similar document given here is this.

(Refer Slide Time: 20:00)

So, all these documents are essentially coming from a single work or a single book and these are different paragraphs of the same article. So, they are more or less similar, still they have difference relative, differences within their similarity.

So, the doc2vec essentially helps in handling the documents similarity between documents in a given corpus. So, this will be this is all that we will be discussing regarding distribution similarity, or distribution semantics especially using word vectors.

Thank you.